
TIP845-SW-82

Linux Device Driver

48 Channel 14-bit A/D Converter

Version 1.0.x

User Manual

Issue 1.0.0

June 2005

TIP845-SW-82

48 Channel 14-bit A/D Converter

Linux Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	June 07, 2005

Table of Content

1	INTRODUCTION.....	4
2	INTRODUCTION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	6
	2.3 Install device driver into the running kernel	6
	2.4 Remove device driver from the running kernel	7
	2.5 Change Major Device Number	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	10
	3.3 ioctl()	11
	3.3.1 T845_IOCTL_READ_ADC	13
	3.3.2 T845_IOCTL_READ_SEQ	15
	3.3.3 T845_IOCTL_START_SEQ	18
	3.3.4 T845_IOCTL_STOP_SEQ	21

1 Introduction

The TIP845-SW-82 Linux device driver allows the operation of a TIP845 14-bit A/D Converter IPAC module on Linux operating systems.

Because the TIP845 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP845 device driver includes the following features:

- Read ADC data in normal mode
- Read ADC data in sequencer mode
- Use of gain 1, 2, 4 or 8 for every channel
- Selection of single-ended or differential input interface
- Optional data correction with factory stored calibration data
- Selectable sequencer cycle time

2 Introduction

The directory TIP845-SW-82 on the distribution media contains the following files:

TIP845-SW-82.pdf	This manual in PDF format
TIP845-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TIP845-SW-82.tar.gz contains the following files and directories:

tip845/tip845.c	Driver source code
tip845/tip845def.h	Driver include file
tip845/tip845.h	Driver include file for application program
tip845/tpmodule.c	Driver independent library
tip845/tpmodule.h	Driver independent library header file
tip845/makenode	Script to create device nodes on the file system
tip845/Makefile	Device driver make file
tip845/example/tip845example.c	Example application
tip845/example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP845-SW-82.tar.gz to the desired target directory.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-82* on the distribution media.

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

make install

- Also after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by *modprobe* to automatically load the correct IPAC carrier driver modules.

depmod -aq

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

```
# make uninstall
```

- Update kernel module dependency description file

```
# depmod -aq
```

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

```
# modprobe tip845drv
```

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled the new device file system (devfs) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

```
# sh makenode
```

On success the device driver will create a minor device for each TIP845 module found. The first TIP845 can be accessed with device node */dev/tip845_0*, the second TIP845 or the second channel of the first TIP845 with device node */dev/tip845_1* and so on.

The allocation of device nodes to physical TIP845 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP845 device driver will only work if kernel KMOD support is installed, necessary carrier board drivers already installed and the kernel dependency file is up to date. If KMOD support isn't available you have to build either a new kernel with KMOD installed or you have to install the IPAC carrier kernel modules manually in the correct order (please refer to the IPAC carrier driver user manual).

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

```
# modprobe tip845drv -r
```

If your kernel has enabled devfs, all /dev/tip845_... nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip845drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe -r* again.

2.5 Change Major Device Number

The TIP845 driver use dynamic allocation of major device numbers by default. If this isn't suitable for the application it's possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TIP845_MAJOR.

To change the major number edit the file tip845drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP845_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP845_MAJOR                      122
```

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

```
#include <fcntl.h>
```

```
int open (const char *filename, int flags)
```

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;  
  
...  
  
fd = open("/dev/tip845_0", O_RDWR);
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV

The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

```
#include <unistd.h>
```

```
int close (int filedes)
```

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

...

if (close(fd) != 0) {
    /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

<i>ENODEV</i>	The requested minor device does not exist.
---------------	--

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

```
#include <sys/ioctl.h>
```

```
int ioctl(int filedes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tip845.h*:

Symbol	Meaning
<i>T845_IOCTL_READ_ADC</i>	Execute conversion and read value
<i>T845_IOCTL_READ_SEQ</i>	Read sequencer data set
<i>T845_IOCTL_START_SEQ</i>	Configure and start sequencer
<i>T845_IOCTL_STOP_SEQ</i>	Stop sequencer

See behind for more detailed information on each control code.

To use these TIP845 specific control codes the header file tip845.h must be included in the application.

RETURNS

On success, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL

Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument request.

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP845 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 T845_IOCTL_READ_ADC

NAME

T845_IOCTL_READ_ADC – Read ADC input value

DESCRIPTION

This ioctl function starts and reads the result of an AD conversion of a specified channel from the TIP845 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp*.

The parameter buffer (*T845_ADC_BUFFER*) has the following layout:

```
typedef struct
{
    int          channel;
    int          gain;
    unsigned long flags;
    short        value;
} T845_ADC_BUFFER, *PT845_ADC_BUFFER;
```

channel

This argument specifies the channel number that will be used. Allowed channel numbers for single-ended input channels are 1 up to 48, for differential input channels allowed channel numbers are 1 up to 24.

gain

This parameter specifies the input gain. Allowed values are 1, 2, 4, and 8.

flags

This parameter contains flags specifying the input interface and if data correction shall be used or not. The flags must be ORed. The following flags are valid for this function:

Flag	Description
<i>T845_FLAG_ADC_CORR</i>	If this flag is set the input value will be corrected using the factory stored correction data. If this flag is unset, the value will be untouched.
<i>T845_FLAG_ADC_DIFF</i>	This flag selects differential input interface. If this flag is not set, the input interface will be single-ended.

value

This parameter returns ADC input value (corrected or uncorrected). This value is a sign extended 14-bit value. The value range is from -8192 to 8191.

EXAMPLE

```
int fd;
int result;
T845_ADC_BUFFER adcBuf;

...

/* Make an A/D conversion on differential channel 4 */
/* use gain 2 and make no data correction */
adcBuf.channel = 4;
adcBuf.gain = 2;
adcBuf.flags = T845_FLAG_ADC_DIFF;

result = ioctl(fd, T845_IOCTL_READ_ADC, &adcBuf);
if (result >= 0)
{
    /* OK */
    printf("Value: %d\n", adcBuf.value);
}
else
{
    /* ERROR */
}

...
```

ERRORS

<i>EFAULT</i>	Invalid pointer to the buffer.
<i>EBUSY</i>	The module is in use, sequencer mode is activated.
<i>EINVAL</i>	Invalid flags specified
<i>ECHRNG</i>	Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.2 T845_IOCTL_READ_SEQ

NAME

T845_IOCTL_READ_SEQ – Reads sequencer input data

DESCRIPTION

This ioctl function waits for new data and reads the data from sequencer data RAM of the TIP845 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp*.

The parameter buffer (*T845_SEQ_READ_BUFFER*) has the following layout:

```
typedef struct
{
    short          value[48];
    unsigned long  status;
} T845_SEQ_READ_BUFFER, *PT845_SEQ_READ_BUFFER;
```

value[]

This array is used to return the read sequencer data. The data assigned an array index with a channel dependent value. To get the right index, a MACRO *T845_SEQ_IDX(diff,chan)* is defined in *tip845.h*.

For differential input channels you should address the array as follows:

```
buf.value[T845_SEQ_IDX(TRUE, <n>)]
    TRUE          specifies differential input
    <n>           is the channel number (1...24)
```

For single-ended input channels you should address the array as follows:

```
buf.value[T845_SEQ_IDX(FALSE, <n>)]
    FALSE         specifies differential input
    <n>           is the channel number (1...48)
```

status

This parameter returns the sequencer state. If the sequencer has been stopped on error, this value will contain status flags. The following status flags are defined:

Flag	Description
T845_STATUS_IRAM_ERR	This flag signals that the sequencer is stopped on an instruction RAM error.
T845_STATUS_TIMER_ERR	This flag signals that the sequencer is stopped on an timer error.
T845_STATUS_OVERFLOW	This flag signals that the sequencer found a data overrun error.

If any status flag is set the returned data values may not be valid.

EXAMPLE

```
int fd;
int result;
T845_SEQ_READ_BUFFER seqRdBuf;

...

/* Read sequencer data */

result = ioctl(fd, T845_IOCTL_READ_SEQ, &seqRdBuf);
if (result >= 0)
{
    if (seqRdBuf.status)
    {
        /* Sequencer stopped by error */
    }
    else
    {
        /* OK */
        printf("#1 (sngl): %d\n", seqRdBuf.value[T845_SEQ_IDX(FALSE,1)]);
        printf("#2 (diff): %d\n", seqRdBuf.value[T845_SEQ_IDX(TRUE,2)]);
    }
}
else
{
    /* ERROR */
}

...
```

ERRORS

EFAULT

EBUSY

ETIME

Invalid pointer to the read buffer

The module is not in sequencer mode

The read request timed out

SEE ALSO

ioctl man pages

3.3.3 T845_IOCTL_START_SEQ

NAME

T845_IOCTL_START_SEQ – Configures and starts the sequencer

DESCRIPTION

This ioctl function configures and starts the AD conversion sequencer of the TIP845 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp*.

The parameter buffer (*T845_SEQ_START_BUFFER*) has the following layout:

```
typedef struct
{
    T845_SEQ_SETUP_CHAN    snglChanConf[48];
    T845_SEQ_SETUP_CHAN    diffChanConf[24];
    unsigned short         seqTimer;
    long                   seqReadTimeout;
} T845_SEQ_START_BUFFER, *PT845_SEQ_START_BUFFER;
```

snglChanConf[]

This array contains the sequencer information for single-ended input channels. The index specifies the input channel, 0 for channel 1, 1 for channel 2 and so on.
This is an array of a structure *T845_SEQ_SETUP_CHAN* that is described below.

diffChanConf[]

This array contains the sequencer information for differential input channels. The index specifies the channel, 0 for channel 1, 1 for channel 2 and so on.
This is an array of a structure *T845_SEQ_SETUP_CHAN* that is described below.

seqTimer

This parameter specifies the sequencer cycle time. The time is specified in steps of 100µs. A value of 0 selects the continuous mode. (Refer to the TIP845 User Manual)

seqReadTimeout

This parameter specifies the maximum time to wait for sequencer data. The value is specified in ticks.

Single-ended and differential channels are using the same input lines. It is not possible to use input lines for single-ended and differential at the same time. (Please refer to the TIP845 User manual for a description of the input line assignment.)

The parameter buffer (*T845_SEQ_SETUP_CHAN*) has the following layout:

```
typedef struct
{
    int          gain;
    unsigned long flags;
} T845_SEQ_SETUP_CHAN, *PT845_SEQ_SETUP_CHAN;
```

gain

This value specifies the input gain that should be used for the associated channel. Valid gains are 1, 2, 4, and 8.

flags

This parameter specifies contains flags specifying if data correction shall be used or not and if the associated channel shall be used or not. The flags must be ORed. The following flags are valid for this function:

Flag	Description
<i>T845_FLAG_ADC_CORR</i>	If this flag is set the input value will be corrected using the factory stored correction data. If this flag is unset, the value will be untouched.
<i>T845_FLAG_ADC_ENABLE</i>	If this flag is set the associated channel will used in sequencer mode, if no set the channel will not be used.

EXAMPLE

```
int fd;
int result;
T845_SEQ_START_BUFFER seqBuf;
int i;

...

/* Start sequencer with: */
/* cycle time: 1 sec */
/* timeout: 2000 ticks */
/* Use single-ended channels: 1,2 */
/* Use differential channels: 8,9 */
seqBuf.seqTimer = 10000;
seqBuf.seqReadTimeout = 2000;

for (i = 0; i < 48; i++) /* Initialize single-ended array */
    seqBuf.snglChanConf[i].flags = 0;

/* Single-Ended Channel 1: gain=2, no data correction */
seqBuf.snglChanConf[0].gain = 2;
seqBuf.snglChanConf[0].flags = T845_FLAG_ADC_ENABLE;

...
```

```
...

/* Single-Ended Channel 2: gain=8, data correction */
seqBuf.snglChanConf[1].gain = 8;
seqBuf.snglChanConf[1].flags = T845_FLAG_ADC_ENABLE | T845_FLAG_ADC_CORR;

for (i = 0; i < 24; i++) /* Initialize differential array */
    seqBuf.diffChanConf[i].flags = 0;

/* Differential Channel 8: gain=1, no data correction */
seqBuf.snglChanConf[7].gain = 1;
seqBuf.snglChanConf[7].flags = T845_FLAG_ADC_ENABLE;

/* Differential Channel 9: gain=4, data correction */
seqBuf.snglChanConf[8].gain = 4;
seqBuf.snglChanConf[8].flags = T845_FLAG_ADC_ENABLE | T845_FLAG_ADC_CORR;

result = ioctl(fd, T845_IOCTL_START_SEQ, &seqBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}

...
```

ERRORS

<i>EFAULT</i>	Invalid pointer to the buffer.
<i>EBUSY</i>	Sequencer mode is already activated or enabled channels (single-ended/differential) share an input line.
<i>EINVAL</i>	Invalid flags or gain specified

SEE ALSO

ioctl man pages

3.3.4 T845_IOCTL_STOP_SEQ

NAME

T845_IOCTL_STOP_SEQ – Stops the sequencer

DESCRIPTION

This ioctl function stops the sequencer of the TIP845 associated with the open file descriptor, *filedes*. No parameter buffer is needed, *argp* should be set *NULL*.

EXAMPLE

```
int fd;
int result;

...

/* Stop sequencer */
result = ioctl(fd, T845_IOCTL_STOP_SEQ, NULL);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}

...
```

ERRORS

EBUSY

The sequencer mode is not active.

SEE ALSO

ioctl man pages